Journal of Mathematical Modelling and Algorithms **3:** 367–402, 2004.

The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms

TOMMASO SCHIAVINOTTO and THOMAS STÜTZLE
Darmstadt University of Technology, Computer Science Department, Hochschulstr. 10, 64289 Darmstadt, Germany.
e-mail: {schiavin,tom}@intellektik.informatik.tu-darmstadt.de

Abstract. The linear ordering problem is an $\mathcal{NP}$-hard problem that arises in a variety of applications. Due to its interest in practice, it has received considerable attention and a variety of algorithmic approaches to its solution have been proposed. In this paper we give a detailed search space analysis of available benchmark instance classes that have been used in various researches. The large fitness-distance correlations observed for many of these instances suggest that adaptive restart algorithms like iterated local search or memetic algorithms, which iteratively generate new starting solutions for a local search based on previous search experience, are promising candidates for obtaining high performing algorithms. We therefore experimentally compared two such algorithms and the final experimental results suggest that, in particular, the memetic algorithm is a new state-of-the-art approach to the linear ordering problem.

Mathematics Subject Classifications (2000): 90C27, 90C59, 68W40.

Key words: linear ordering problem, search space analysis, benchmark instances, iterated local search, memetic algorithms.

1. Introduction

Given an $n \times n$ matrix C, the linear ordering problem (LOP) is the problem of finding a permutation π of the column and row indices $\{1, \ldots, n\}$ such that the value

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{\pi_i \pi_j}$$

is maximized. In other words, the goal is to find a permutation of the columns and rows of C such that the sum of the elements in the upper triangle is maximized.

The LOP arises in a large number of applications in a number of fields such as economy, sociology, graph theory, archaeology, and task scheduling [10]. Two well known examples of the LOP are the triangularization of input–output matrices of an economy, where the optimal ordering allows economists to extract some information about the stability of the economy, and the stratification problem in archaeology, where the LOP is used to find the most probable chronological order

of samples found in different sites. The matrix that describes the latter problem is known as Harris Matrix.

The LOP is $\mathcal{NP}$-hard, that is, we cannot expect to find a polynomial time algorithm for its solution. However, since the LOP arises in a variety of practical applications, algorithms for its efficient solution are required. Therefore, several exact and heuristic algorithms were proposed in the literature. Exact algorithms include a branch & bound algorithm that uses a LP-relaxation of the LOP for the lower bound by Kaas [15], a branch & cut algorithm proposed by Grötschel, Jünger, and Reinelt [10] and a combined interior point/cutting plane algorithm by Mitchell and Borchers [23]. State-of-the-art exact algorithms can solve fairly large instances from specific instance classes with up to a few hundred columns and rows, while they fail on much smaller instances of other classes. Independent of the type of instances solved, the computation time of exact algorithms increases strongly with instance size.

The LOP was also tackled by a number of heuristic algorithms. These include constructive algorithms like Becker's greedy algorithm [3], local search algorithms like the $\mathcal{CK}$ heuristic by Chanas and Kobylanski [8], as well as a number of metaheuristic approaches such as elite tabu search and scatter search, presented in a series of papers by Martí, Laguna, and Campos [6, 7, 18], or iterated local search (ILS) algorithms [9, 25]. In particular, ILS approaches appear currently to be the most successful metaheuristics, as judged by their performance on a number of available LOP benchmark instances [9, 25].

Available algorithms have typically been tested on a number of classes of real-world as well as randomly generated instances. However, little is known about how the performance of current state-of-the-art algorithms depends on specific characteristics of the various available LOP instance classes neither how differences among the instances translate into differences in their search space characteristics. First steps into answering these open questions were undertaken in [25].

The main contributions of this article are the following. First, we give a detailed analysis of the search space characteristics of all the instance classes introduced in the major algorithmic contributions to the LOP. This includes a structural analysis, where standard statistical information is gathered as well as an analysis of the main search space characteristics such as autocorrelation [27, 33] and fitness-distance analysis [14]. A second contribution is the detailed analysis of two metaheuristics, an iterated local search algorithm [19] and a memetic algorithm [24]. As we will see, their relative performance depends on the particular LOP instance class to which they are applied. Interestingly, a detailed analysis shows that there is some correlation between specific search space characteristics and the hardness of the instances as encountered by the two metaheuristics. Computational comparisons of the two algorithms to known metaheuristic approaches establish that the memetic algorithm is a new state-of-the-art algorithm for the LOP.

The paper is structured as follows. Section 2 gives an overview of the instance classes that we studied. Section 3 introduces the greedy algorithm and the local

search techniques that are used by the metaheuristics. Details on the structural analysis of the benchmark instances is given in Section 4 and Section 5 describes the results of the search space analysis. Section 6 introduces the metaheuristics we applied and Section 7 gives a detailed account of the computational results. We conclude in Section 8.

2. LOP Instance Classes

So far, research on the LOP has made use of a number of different classes of benchmark instances, including instances stemming from real-world applications as well as randomly generated instances. However, typically not all the instances are tackled in all the available papers and, in addition, several of the randomly generated instances are not available publically.

The probably most widely used class of instances are those of LOLIB, a benchmark library for the LOP that comprises 49 real-world instances corresponding to input–output tables of economical flows in the EU. LOLIB is available at `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/LOLIB/`; for all LOLIB instances optimal solutions are known [11].

Our initial results with an ILS algorithm for the LOP, which were presented in [25], indicated that the LOLIB instances are actually too small to pose a real challenge to state-of-the-art metaheuristic approaches and also to exact algorithms. Therefore, we have generated an additional set of large, random, real-life like instances through sampling uniformly at random elements from the corresponding original LOLIB instances. We call this instance class XLOLIB for eXtended LOLIB. Due to the process used for generating them, XLOLIB instances have a structure similar to those of LOLIB. We generated for each instance of LOLIB two instances, one of size $n = 150$ and another one of size $n = 250$, resulting in a total of 49 instances of size 150 and 49 instances of size 250. Initial tests showed that these instances are well beyond the capabilities of the exact algorithm by Mitchell and Borchers [23], one of the best performing exact algorithms. For example, on one instance of size 250 we aborted the program after one week computation time without identifying an optimal solution.

A real life instance consisting of a single input–output matrix of 81 sectors of the U.S. economy is available from the Stanford graph-base library [17] that is accessible at `http://www-cs-faculty.stanford.edu/~knuth/sgb.html`. From this instance, we generated eight additional smaller instances by randomly selecting sectors: three instances for each of $n \in \{50, 65\}$, one with dimension 79 and one with dimension 80 (plus the original one of dimension 81). We refer to this class with SGB. Instances that were generated in an analogous way have been used to evaluate tabu search and iterated local search algorithms [18, 9].

Because LOLIB instances are rather small, Mitchell and Borchers [23] randomly generated large instances in their research on exact algorithms for the LOP. They generated a matrix by first drawing numbers between 0 and 99 for the ele-

ments in the upper triangle, and between 0 and 39 for the others, and then shuffling the matrix. This technique is used in order to obtain a linearity similar to the LOLIB instances; the linearity is the ratio of the optimal objective function value over the sum of the matrix elements excluding those on the diagonal. Furthermore, zeros are added to increase the sparsity of the matrix. The range used to draw numbers is extremely limited when compared with the values that LOLIB instances elements can take. The idea underlying this choice is that there should be a large number of solutions with costs close to the optimal value, which, according to Mitchell and Borchers, yields hard instances. Thirty of these instances with known optimal solutions are available at `http://www.rpi.edu/~mitchj/generators/linord`, where also the generator can be found; we will refer to these instances as MBLB (Mitchell–Borchers LOP Benchmarks). Of these 30 instances, five are of size 100, ten of size 150, ten of size 200, and five of size 250. Even if the size of the MBLB instances is comparable to those of XLOLIB, preliminary tests showed that MBLB instances are significantly easier than XLOLIB instances. In fact, for all MBLB instances provably optimal solutions are known.

Finally, another class of randomly generated instances was proposed by Laguna, Martí, and Campos [18]. They generated 25 instances for each of $n \in \{75, 150, 200\}$ resulting in a total of 75 instances. The matrix entries are generated according to a uniform distribution in $\{0, 1, \ldots, 250\,000\}$. We call this instance class LMC-LOP. These instances were made available by Rafael Martí.

3. Constructive and Local Search Algorithms

The currently best known constructive algorithm for the LOP is due to Becker [3]. In a first step the index that maximizes the cost

$$q_i = \frac{\sum_{k=1}^{n} c_{ik}}{\sum_{k=1}^{n} c_{ki}}, \quad i = 1, \ldots, n$$

is chosen, and it is put in the first position of the permutation. Next, this index together with the corresponding column and row is removed and the new q_i values for the remaining indices are computed from the resulting sub-matrix. These steps are repeated until the index list is empty, resulting in a computational cost of $\mathcal{O}(n^3)$. A variation of this algorithm is to compute the q_i values only once at the start of the algorithm, sort these values in non-increasing order to yield a permutation of the indices. Using this variant, a solution can be computed in $\mathcal{O}(n^2)$.

Both, the original algorithm and the variation, return good quality solutions compared to random ones. For example, the average deviation from optimal solution quality for LOLIB instances is 6.52% with the original algorithm, 9.46% with the variation, and 30.48% for random solutions; for MBLB the deviations obtained are 2.91% with the original algorithm, 2.52% with the static variation, and 40.34% for random solutions.

Better solutions than with Becker's construction heuristic are obtained with local search algorithms. We considered three possibilities. The first two are based on neighborhoods defined through the operations applicable to the current solution.

The first neighborhood, $\mathcal{N}_X$, is defined by the operation *interchange*: $\Pi(n) \times \{1, \ldots, n\}^2 \to \Pi(n)$, where $\Pi(n)$ is the set of all permutations of $\{1, \ldots, n\}$ and we have for $i \neq j$:

$$interchange(\pi, i, j) \triangleq (\pi_1, \ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_{j+1}, \ldots, \pi_n).$$

The $\mathcal{N}_X(\pi)$ (the neighborhood of a permutation π), is defined to be the set of all permutations that can be obtained by applying one interchange move to π. This neighborhood has, for any π, size $|\mathcal{N}_X(\pi)| = n(n-1)/2$ and a complete neighbourhood evaluation has cost $\mathcal{O}(n^3)$. Preliminary tests showed that $\mathcal{N}_X$ gives significantly worse solution quality when compared to the following two local search methods.

A second neighborhood, $\mathcal{N}_I$, is defined by the *insert* operation: an element in position i is inserted in another position j. Formally, *insert*: $\Pi(n) \times \{1, \ldots, n\}^2 \to \Pi(n)$ is defined for $i \neq j$:

$$insert(\pi, i, j) \triangleq \begin{cases} (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_j, \pi_i, \pi_{j+1}, \ldots, \pi_n), & i < j; \\ (\pi_1, \ldots, \pi_{j-1}, \pi_i, \pi_j, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n), & i > j. \end{cases}$$

The insert based neighborhood has size $|\mathcal{N}_I(\pi)| = (n-1)^2$.

The Δ-function

$$\Delta_I(\pi, i, j) \triangleq f(insert(\pi, i, j)) - f(\pi)$$

associated with this operation is defined as:

$$\Delta_I(\pi, i, j) \triangleq \begin{cases} \sum_{k=i+1}^{j} c_{\pi_k \pi_i} - c_{\pi_i \pi_k}, & i < j; \\ \sum_{k=j}^{i-1} c_{\pi_i \pi_k} - c_{\pi_k \pi_i}, & i > j. \end{cases}$$

The cost for evaluating this function is $\mathcal{O}(|i-j|)$ and, hence, in $\mathcal{O}(n)$ if no care is taken. In a straightforward implementation of a local search based on this neighborhood one would, given an index i, tentatively try all possible moves $insert(\pi, i, j)$ with j ranging from 0 to $n-1$. Since for each move the Δ-function evaluation is linear, the cost for exhaustively visiting the neighborhood is $\mathcal{O}(n^3)$. However, the local search procedure can be sped up significantly, if the neighborhood is visited in a more systematic way. A particular case of an insert move is given if $i = j \pm 1$; we call this a *swap* move, that is $swap(\pi, i) = (\pi_1, \ldots, \pi_{i+1}, \pi_i, \ldots, \pi_n)$, and its Δ function is:

$$\Delta_S(\pi, i) \triangleq c_{\pi_{i+1}\pi_i} - c_{\pi_i \pi_{i+1}}.$$

Hence, the cost of the evaluation of Δ_S is constant. Furthermore, an *insert* move (with arguments i and j) is always equivalent to a sequence of $|i-j|$ *swap* moves.

For example, for $n = 6$ the insert move with $i = 2$, $j = 5$ can be written as a sequence of three swap moves:

$$\begin{aligned} \pi &= 1\,2\,3\,4\,5\,6 && \rightarrow \\ \mathit{swap}(1\,2\,3\,4\,5\,6, 2) &= 1\,3\,2\,4\,5\,6 = \mathit{insert}(\pi, 2, 3) && \rightarrow \\ \mathit{swap}(1\,3\,2\,4\,5\,6, 3) &= 1\,3\,4\,2\,5\,6 = \mathit{insert}(\pi, 2, 4) && \rightarrow \\ \mathit{swap}(1\,3\,4\,2\,5\,6, 4) &= 1\,3\,4\,5\,2\,6 = \mathit{insert}(\pi, 2, 5). \end{aligned}$$

In the example, it can be noticed that all the permutations visited are in the *insert*-neighborhood of π. Hence, the idea is to use only *swap*-moves to visit the whole *insert*-neighborhood. For each index i we will apply all the possible moves $\mathit{insert}(\pi, i, j)$ in two stages. First, with indices j that range from $i - 1$ to 0 and then for indices j that vary from $i + 1$ to $n - 1$. In every stage a solution can be obtained from the previous visited one by applying a *swap* move. Hence, every solution in the neighborhood can be obtained in constant time and therefore the total computational cost for evaluating the insert neighborhood becomes $\mathcal{O}(n^2)$. This technique was inspired by the method that Congram applies to Dynasearch [9]. Figure 1 shows the effect of an *insert* move on the matrix and how the same move can be done only through *swap* moves.

A further, minor speed-up consists in pre-computing the cost for all the possible *swap* moves:

$$d_{ij} = c_{ij} - c_{ji} \quad \forall i, j = 0, \ldots, n - 1.$$

In addition to these standard neighborhoods, we also implemented the $\mathcal{CK}$ local search algorithm by Chanas and Kobylański [8] that uses two functions *sort* and *reverse*. When applied to a permutation, *sort* returns a new permutation in which the elements are rearranged according to a specific sorting criterion (see [8]), while *reverse* returns the reversed permutation. In the LOP case, if a permutation maximizes the objective function, the reversed permutation minimizes the objective function; hence, reversing a good solution leads to a bad solution. The idea of $\mathcal{CK}$ is to alternate sorting and reversing to improve the current solution; in fact, it has been shown that the application of *reverse* and then *sort* to a solution will lead to a solution with a value greater or equal to the starting one. The functional description of the algorithm is:

$$(\mathit{sort}^* \circ \mathit{reverse})^* \circ \mathit{sort}^*,$$

where $\circ$ denotes function composition, and the * operator is used to apply any given function iteratively until the objective function does not change. Formally, we consider a general function ϕ, and a generic permutation π:

$$\phi^*(\pi) \triangleq \begin{cases} \pi, & f(\phi(\pi)) = f(\pi), \\ \phi^*(\phi(\pi)), & \text{otherwise}. \end{cases}$$

The sort function is recursively defined as follows:

$$\mathit{sort}(\pi_1, \ldots, \pi_k) = \begin{cases} \pi_1, & k = 1, \\ \mathit{insert}_{\mathcal{CK}}(\pi_k, \mathit{sort}(\pi_1, \ldots, \pi_{k-1})), & k > 1, \end{cases} \tag{1}$$

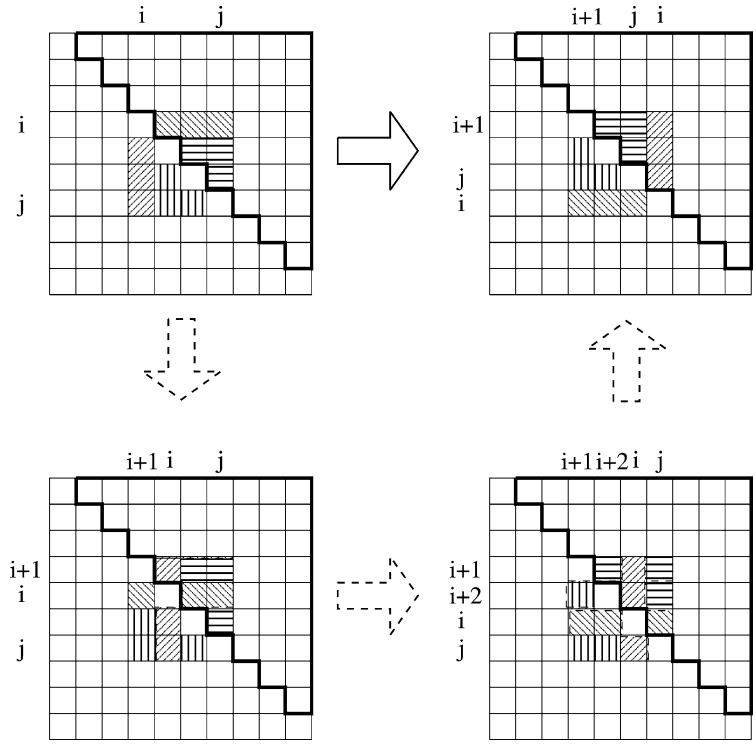

Figure 1. Given is a pictorial example of what the Δ function means in terms of matrix entries and how an insert move can be done with successive *swap* moves. The shadings of the cells visually illustrate for given i and j how the objective function is affected by the single swap moves. The insert move swaps a group of cells on the upper triangular with another on the lower (upper part). The same operation can be obtained by a sequence of swap operations (indicated by the interlined arrows). Notice that when a swap is applied, the cells related to the swapped indices change position, but only the two close to the diagonal pass from the upper triangular to the lower and vice versa.

$$insert_{\mathcal{CK}}(t, (\pi_1, \ldots, \pi_{k-1})) = (\pi_1, \ldots, \pi_{\bar{r}-1}, t, \pi_{\bar{r}}, \ldots, \pi_{k-1}), \tag{2}$$

where $\bar{r} \in \{1, \ldots, k\}$ such that it maximizes the value:

$$\Delta_{\mathcal{CK}}(t, r, (\pi_0, \ldots, \pi_{k-1})) = \sum_{j=1}^{\bar{r}-1} c_{\pi_j t} + \sum_{j=\bar{r}}^{k-1} c_{t\pi_j}.$$

Unfortunately, this definition does not help in understanding the neighborhood actually used by $\mathcal{CK}$. In fact, one can show that the $\mathcal{CK}$ algorithm actually implements a local search algorithm based on $\mathcal{N}_I$.

We implemented three local search variants, including two versions of the *insert* moves and $\mathcal{CK}$. The two *insert* variants differ only in the pivoting rule applied. One version uses a pivoting rule that is between *first* and *best improvement*:

Table I. Comparison between three local search algorithms on the benchmark classes. The results are averaged over all instances of each class and over 100 trials for each instance. Avg.Dev. gives the average percentage deviation from optimal or best known solutions, # optima gives the number of optimal or best known solutions found at least once in the 100 trials for each instance, and Avg.time (sec) gives the average computation time in seconds on a 1400+ Athlon CPU to run the local search once on each benchmark instance of a class (for example, the timing given on LOLIB instances is the time to run a local search once for all the 49 instances of LOLIB)

		Avg.Dev. (%)	# optima	Avg.time (sec)
LOLIB	$\mathcal{LS}_i$	0.1842	42	0.1802
	$\mathcal{CK}$	0.2403	38	0.0205
	$\mathcal{LS}_f$	0.22	45	0.013
SGB	$\mathcal{LS}_i$	0.27	5	0.1389
	$\mathcal{CK}$	0.41	3	0.01013
	$\mathcal{LS}_f$	0.46	7	0.00516
MBLB	$\mathcal{LS}_i$	0.0195	10	9.81
	$\mathcal{CK}$	0.0209	12	0.22
	$\mathcal{LS}_f$	0.021	10	0.14
XLOLIB (250)	$\mathcal{CK}$	1.11	0	2.1256
	$\mathcal{LS}_f$	0.90	0	0.6741
LMC-LOP	$\mathcal{CK}$	0.65	0	1.0496
	$\mathcal{LS}_f$	0.60	0	0.2976

function $visit_{\mathcal{N}_I}(\pi)$
  for $i = 0 \,..\, n-1$ **do**
    $\bar{r} \leftarrow \arg\max_{r.r \neq i} f(insert(\pi, i, r))$
    $\pi' = insert(\pi, i, \bar{r})$
    if $f(\pi') > f(\pi)$ **then**
      return(π')
    end if
end for
return(π)

Obviously, the scan of the indices for finding the best move for each index is made exploiting the evaluation of the delta function in constant time. We indicate with $\mathcal{LS}_f$ the local search based on the visit-function we just introduced; $\mathcal{LS}_i$ is a local search on $\mathcal{N}_I$ using a random first improvement strategy, where the neigh-

borhood is scanned in random order; the latter neighborhood examination scheme requires the Δ-function to be computed in linear time.

Table I gives a comparison of the three algorithms on the benchmark classes we considered. The results show that with respect to solution quality all three algorithms are comparable. However, they strongly differ in terms of computational speed. Clearly, $\mathcal{LS}_f$ is the fastest, followed by $\mathcal{CK}$; $\mathcal{LS}_i$ is several orders of magnitude slower than the other two. Based on these results, in the rest of the paper we will only apply $\mathcal{LS}_f$.

4. Structural Analysis of the Instances

As a first step in our analysis of the LOP instance characteristics, we computed cross-statistical information on the distribution of the matrix entries for the available instances. In particular, we computed for all instances the sparsity, the variation coefficient and the skewness of the matrix entries. The sparsity measures the percentage of matrix elements that are equal to zero; the main interest in this measure is that, according to Mitchell and Borchers, it has a strong influence on the behaviour of their algorithm [23]. The *variation coefficient* (VC) is defined as $\sigma/\bar{x}$, where σ is the standard deviation and $\bar{x}$ the mean of the matrix entries. VC gives an estimate of the variability of the matrix entries, independent of their size. The *skewness* is the third moment of the mean normalized by the standard deviation; it gives an indication of the degree of asymmetry of the matrix entries. The statistical data is given in Table II for LOLIB and XLOLIB instances and in Table III for the random instance classes MBLB and LMC-LOP.

The cross statistical data for the SGB instances are the following: the median of the sparsity is 14.16 and 22.91 for the size 50 and 65 instances, respectively; the VC is 4.59 and 5.23, respectively; the skewness is 10.31 and 13.09, respectively. For the size 79, 80 and 81 instances the sparsity is 26.26, 25.91, and 25.29, respectively; the VC is 6.12, 6.13, and 10.62, respectively; and the skewness is 16.62, 16.80, and 21.27, respectively.*

These statistical data show that there are significant differences between the real-life instances (LOLIB and SGB) and real-life like random problems (XLOLIB) on the one side and the randomly generated instances from LMC-LOP and MBLB on the other side. For the real-life and real-life like instances all statistics (sparsity, VC, and skewness) are typically much higher than for the randomly generated instances. This suggests that the former class of instances are much less regular and the variation among the matrix entries is much stronger than for the random instances. Additionally, the variation of the statistics among the real-world instances is much larger indicating a certain diversity of structural features in these instances. Obviously, SGB instances are an exception in that respect, because they

* Note that the full SGB instance of size 81 has the particularity of having a negative row, hence resulting in a somewhat different structure of this instance with respect to VC and skewness than other instances of similar size.

Table II. Structural information on the real-world and real-world like instance classes. "Sp." indicates the sparsity, "VC" the variation coefficient, and "Sk." the skewness. Given are the minimum, the 0.25 and 0.75 quantiles, the median, the maximum and the mean of these measures across all instances of a benchmark class

				LOLIB			
Size		Min	1st qu.	Median	3rd qu.	Max	Mean
	Sp.	11.00	26.91	35.28	46.13	80.63	37.34
all	VC	4.10	4.45	4.87	5.78	16.25	5.49
	Sk.	9.15	11.40	12.93	15.83	39.18	15.50
				XLOLIB			
	Sp.	10.57	26.80	34.71	45.74	80.351	37.25
150	VC	4.04	4.46	4.84	5.54	16.05	5.42
	Sk.	8.94	11.09	12.49	15.78	42.62	15.04
	Sp.	10.79	26.98	35.04	45.76	80.48	37.25
250	VC	4.07	4.39	5.00	5.77	15.81	5.48
	Sk.	9.05	11.33	12.61	16.51	43.63	15.49

are all generated from the same matrix. Differently, the variance of the statistical measures is low for LMC-LOP and MBLB instances, indicating a more regular structure of these.

The data presented here give evidence that we might observe significant differences in the behavior of algorithms when applied to random instances or real-life (like) instances. Additionally, these data give an indication that conclusions obtained for the random instances do not necessarily apply to real-life instances, because random instances show very different statistical data from real-life instances. Hence, the XLOLIB instances appear to be much better suited for testing algorithms on realistic, large LOP instances than the random instances.

5. Landscape Analysis

Central to the landscape analysis in combinatorial optimization is (i) a representation of the space searched by an algorithm as a landscape formed by all feasible solutions, which in the LOP case are permutations, (ii) a *fitness* value assigned to each solution, which in our case is the objective function value $f(\pi)$ of a permutation π, and (iii) a distance metric on the search space [21]. The usefulness of landscape analysis is typically based on the insights with respect to landscape characteristics and the relationship to the behavior of local search algorithms or metaheuris-

Table III. Structural information on randomly generated instance classes. "Sp." indicates the sparsity, "VC" the variation coefficient, and "Sk." the skewness. Given are the minimum, the 0.25 and 0.75 quantiles, the median, the maximum and the mean of these measures across all instances of a benchmark class

LMC-LOP							
Size		Min	1st qu.	Median	3rd qu.	Max	Mean
	Sp.	0.5	0.5	0.50	0.51	0.51	0.51
75	VC	0.70	0.71	0.71	0.71	0.72	0.71
	Sk.	0.389	0.40	0.40	0.40	0.42	0.40
	Sp.	1.33	1.33	1.33	1.33	1.37	1.34
150	VC	0.70	0.71	0.72	0.72	0.73	0.72
	Sk.	0.36	0.40	0.41	0.43	0.46	0.41
	Sp.	0.67	0.67	0.67	0.68	0.68	0.67
200	VC	0.71	0.71	0.71	0.71	0.72	0.71
	Sk.	0.38	0.34	0.40	0.41	0.43	0.40
MBLB							
	Sp.	22.01	22.12	22.33	22.44	23.36	22.45
100	VC	1.00	1.01	1.01	1.01	1.02	1.01
	Sk.	0.98	0.98	1.00	1.00	1.00	0.99
	Sp.	2.29	2.38	7.15	12.02	12.48	7.23
150	VC	0.77	0.78	0.83	0.88	0.89	0.83
	Sk.	0.82	0.84	0.86	0.88	0.88	0.86
	Sp.	2.08	2.25	7.02	11.87	12.10	7.06
200	VC	0.77	0.78	0.83	0.88	0.88	0.83
	Sk.	0.82	0.84	0.86	0.88	0.89	0.86
	Sp.	2.04	2.11	2.12	2.15	2.23	2.13
250	VC	0.77	0.77	0.78	0.78	0.78	0.77
	Sk.	0.82	0.83	0.84	0.84	0.84	0.84

tics [5, 30], the possibility to predict problem or problem instance difficulty [2, 28], or indications for useful parameterizations of local search algorithms [1].

Formally, the search landscape of the LOP is described by a triple $\langle \Pi(n), f, d \rangle$, where $\Pi(n)$ is the set of all permutations of the integers $\{1, \ldots, n\}$, f is the objective function and d is a distance measure, which induces a structure on the landscape. It is natural to define the distance between two permutations π and π' so as to depend on the "basic operation" used by a local search algorithm; typically,

the distance is given by the minimum number of applications of this basic operation needed to transform π into π'. Since the best performing local search algorithms are all based on $\mathcal{N}_I$, we consider an *insert* move as our basic operation.

5.1. LANDSCAPE CORRELATION ANALYSIS

The first feature of the search landscape we studied is its ruggedness: a search landscape is said to be rugged if there is a low correlation between neighboring points. To measure this correlation, Weinberger suggested to perform a *random walk* in the search landscape of length m, to interpret the resulting set of m points $\{f(x_t)\}, t = 1, \ldots, m$ as a time series and to measure the autocorrelation $r(s)$ of points in this time series that are separated by s steps [33] as

$$r(s) = \frac{\sum_{t=1}^{m-s}(f(x_t) - \bar{f})(f(x_{t+s}) - \bar{f})}{\hat{\sigma}^2(f)(m-s)},$$

where $\hat{\sigma}^2(f)$ is the variance of the time series, and $\bar{f}$ its mean. Often, the resulting time series can be modeled as an autoregressive process of order one, and then the whole correlation structure can be summarized by $r(1)$ or, equivalently, by the *search landscape correlation length* that is computed as $\ell = -\frac{1}{\ln(|r(1)|)}$ $(r(1) \neq 0)$ [26, 27, 33]; the lower is the value of ℓ, the more rugged is the landscape. Interestingly, in landscape analysis literature general intuitions and some results suggest that there is a negative correlation between ℓ and the hardness of the problem [2].

We computed ℓ on all benchmark instances with a random walk of one million steps; Table IV summarizes data collected on all instances. ℓ is given normalized by the instance size (n). As we see, each class has a relatively small variance (SGB has one instance that represents an outlier in these data). This means, that the landscape correlation length can characterize specific instance classes. From these instance classes, the MBLB instances have by far the largest normalized correlation length, which would suggest that these instances are also the easiest to solve; in fact, when abstracting from instance size, our experimental results with metaheuristics suggest that this is true. The next smaller ℓ/n is found for the real-life instances from LOLIB and SGB, while the smallest values, on average, are observed for LMC-LOP and XLOLIB instances.

Note that the values of ℓ/n alone are not sufficient to serve as the sole basis for the definition of instance classes. For example, XLOLIB instances showed roughly the same normalized values for ℓ/n as LMC-LOP instances, which would suggest similar behavior. However, both types of instances have widely different characteristics as shown by the data on the distribution of the matrix entries.

From a methodological point of view we were interested on how long a *random walk* should be to obtain a stable estimate of ℓ. Therefore, we measured on all MBLB and XLOLIB instances of size 250 ten times ℓ for different lengths of the random walks. Figure 2 shows the values for ℓ/n resulting from these experiments

Table IV. Given are standard statistical data (minimum, 0.25 and 0.75 quantiles, median, average, and maximum) for the normalized values ℓ/n of the search landscape correlation length measured across all the instances of the available benchmark classes

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	0.7536	0.7907	0.8004	0.8207	0.8403	0.8021
SGB	0.4821	0.8055	0.8163	0.8248	0.8347	0.7810
XLOLIB(100)	0.7094	0.7278	0.7311	0.7416	0.7671	0.7341
XLOLIB(250)	0.7165	0.7327	0.7364	0.7407	0.7524	0.7372
MBLB	0.9339	0.9595	0.9639	0.9707	0.9775	0.9620
LMC-LOP	0.6924	0.7211	0.7312	0.7450	0.7746	0.7332

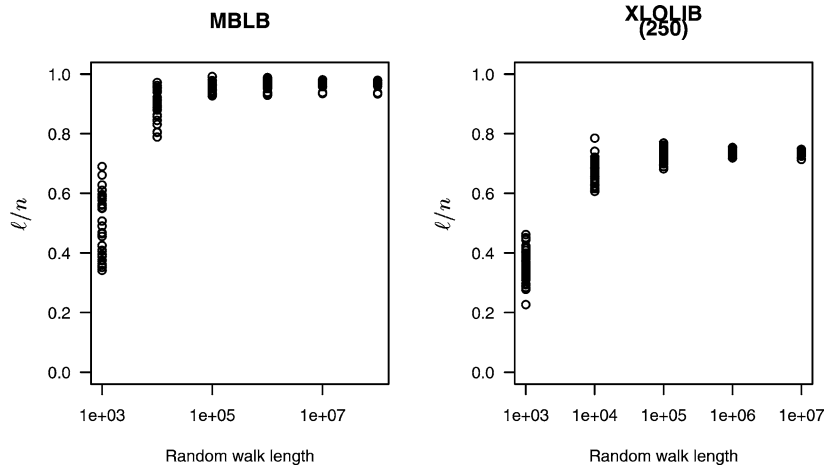

Figure 2. Dependence of ℓ/n (given in the y-axis) on the *random walk* length (x-axis) for MBLB (left) and XLOLIB (right) instances. Every dot gives the average normalized landscape correlation length measured across 10 *random walks* for each instance.

for all instances. As can be seen, the longer is the random walk the more precise is the resulting measure of ℓ. These plots also indicate that apparently for 1,000,000 steps in the random walk the estimate has stabilized. On the other side, these results also suggest that the random walks for measuring ℓ should be by a large multiple (e.g., 400 in this case) larger than the instance size (or the diameter of the landscape, which in this case is $n - 1$) to result in stable estimates.

5.2. FITNESS-DISTANCE ANALYSIS

In a next step we analyzed the distribution and the relative location of local optima to the global optima of the LOP instances.

Table V. Summary information on the percentage of distinct local optima found (percentage of the total number of local optima generated)

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	0.12	40.35	63.65	80.49	99.2	67.93
SGB	25.60	64.50	94.60	99.5	100.00	81.09
MBLB	7.30	76.75	96.50	99.48	100.00	82.17

Table VI. Summary information on the number of distinct global optima found, given as the percentage of the total number of distinct local optima

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	0.47	2.34	14.10	16.54	85.12	8.33
SGB	0.00	0.10	0.20	1.17	1.86	0.68
MBLB	0.00	0.00	0.22	0.57	4.27	0.46

For LOLIB instances we ran 13,000 local searches starting from random solutions, while for the other instance classes 1,000 local searches were done. On the instance classes LMC-LOP and XLOLIB the local searches generated 1,000 distinct local optima for each instance and in no case was the best known solution found. For LOLIB instead, the number of distinct local optima was considerably varying, between 24 and 13,000 with a median around 9,400; for MBLB instances the number ranged from around 73 to 1,000, with a median of 965; finally for the smaller SGB instances around 600 distinct local optima were found, while in the largest ones 1,000 distinct local optima resulted. Summary data are given also in Table V.

For all LOLIB, SGB and MBLB instances we know the global optima. In fact, on several instances we could find global optima among the local optima generated. Among the total number of distinct local optima, the percentage of global optima ranges from 0.47% to 85.12% for LOLIB, while for the other instance classes the corresponding percentages are much smaller. Summary data on these values are given in Table VI. In fact, these results also suggest that especially the LOLIB instances can effectively be solved by a random restart algorithm that is run long enough.

Finally, we analyzed the relationship between the quality of local optima and their distance to the closest global optimum by measuring the fitness distance correlation coefficient and generating fitness distance plots [14]. Given a sample of m candidate solutions $\{\pi^1, \ldots, \pi^m\}$ with an associated set of pairs $\{(f_1, d_1), \ldots,$

Table VII. Some information on the local optima generated, the mean distance from the best known solution is given as percentage over the max distance. The number of distinct local optima is given as percentage of the number of local optima generated; the LOLIB value is over 13,000 trials explaining in part the very low value

Class	Avg.Dist (%) from best known	Avg. (%) distinct local optima
LOLIB	5.84	58.55*
SGB	10.51	81.18
XLOLIB	26.27	100
MBLB	0.43	82.02
LMC-LOP	23.22	100

$(f_m, d_m)\}$ of fitness (solution quality) values f_i and distances to the closest global optimum d_i, the (sample) fitness distance correlation coefficient ρ can be computed as

$$\rho = \frac{\widehat{\mathrm{Cov}}(f,d)}{\hat{\sigma}(f)\cdot\hat{\sigma}(d)}, \tag{3}$$

where

$$\widehat{\mathrm{Cov}}(f,d) = \frac{1}{m-1}\sum_{i=1}^{m}(f_i-\bar{f})(d_i-\bar{d}), \tag{4}$$

$$\hat{\sigma}(f) = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(f_i-\bar{f})^2}, \qquad \hat{\sigma}_F = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(d_i-\bar{d})^2}, \tag{5}$$

and $\bar{f}$, $\bar{d}$ are the averages over the sets $F = \{f_1, \ldots, f_m\}$ and $D = \{d_1, \ldots, d_m\}$, respectively. $\widehat{\mathrm{Cov}}(f,d)$ is the sample covariance between the f and d values, while $\hat{\sigma}_F$ and $\hat{\sigma}_D$ are the sample standard deviations of F and D, respectively. As usual, we have $-1 \leqslant \rho \leqslant 1$. In our case, we used as the fitness the deviation from the global optimum. Hence, a high, positive value of ρ indicates that the higher the solution quality, the closer one gets, on average, to global optima; hence, the solution quality gives good guidance when searching for global optima. For the instances of the classes LMC-LOP and XLOLIB we do not have proven optimal solutions available, since exact algorithms were not able to solve these. In this case, we used the best known solutions instead. The best known solutions were the best ones found by the metaheuristics we tested in Section 7. In the case of the LMC-LOP instances of dimension 75, these best known solutions are conjectured to be the optimal ones, since the same best solutions were found in many trials of the metaheuristics we tested.

In addition, we used a second measure of the FDC that is a variation on the original measure. For this new measure the FDC is computed only for the local

Table VIII. Summary information for ρ, the fitness distance correlation coefficient, computed on the complete set of local optima

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	0.1369	0.5471	0.6057	0.7397	0.8669	0.6173
SGB	0.3408	0.5431	0.5944	0.6222	0.8914	0.5956
XLOLIB	0.2364	0.3903	0.4313	0.4772	0.6243	0.4322
MBLB	0.6516	0.7266	0.7758	0.8344	0.8871	0.7741
LMC-LOP	0.2271	0.4149	0.4824	0.5560	0.8060	0.4850

Table IX. Summary information for ρ', the easy level fitness distance correlation coefficient, computed on the local optima better than the *easy level* (see text for details)

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	0.3087	0.5980	0.7408	0.8612	0.9968	0.7142
SGB	0.4959	0.5655	0.6327	0.7089	0.9386	0.6536
XLOLIB	0.1580	0.2564	0.3145	0.3597	0.5528	0.3143
MBLB	0.4605	0.6539	0.6864	0.7510	0.8394	0.6874
LMC-LOP	0.09149	0.27240	0.33240	0.42820	0.75480	0.35360

optima with an objective function value that is better than the median objective function for all the local optima that were generated. The idea behind this measure is that all the metaheuristics should be able to easily reach a solution with such an objective function value. In fact, by a simple random restart, within a few iterations the probability of finding a solution better than a median local optimum quality approaches one. Furthermore, in an analysis of the FDC relationship one should focus on the solutions which are the more likely ones to be encountered in the search trajectory of the metaheuristics. We will refer to the threshold on the solution quality as the *easy level* and to the FDC based on the solutions passing this bound as the *easy level FDC* (ρ').

Tables VIII and IX summarize the information about FDC and *easy level FDC*, respectively. For the larger instances, it appears that the easy level FDC coefficients are lower than the standard FDC coefficients. This probably is the case because poor quality local optima that are far from the global optimum can have a considerable influence on the resulting correlation and these poor local optima are eliminated when imposing the "easy level" bound. For some instances the values of ρ and ρ' are strongly positive, suggesting that these instances should be relatively easy for restart type algorithms [21].

In Figure 3 we show some example FDC plots for instances from all benchmark classes except XLOLIB. Since the range of the x-axis is from zero to the maximum distance, the plots give visual information on the typical distance of local optima to the nearest global optima (see also Table VII). MBLB instances show the peculiarity that the local optima are relatively close to global optima. In order to give a more detailed picture of the fitness-distance relationship for these instances, we plot in Figure 4 the same data as in Figure 3 but using a logarithmic scale on the x-axis. In Section 7, we will give an analysis of how the fitness distance measure correlates with the hardness of LOP instances for particular metaheuristics.

According to the FDC analysis and the FDC plots, the MBLB instances would be predicted to be the easiest ones among the large instances. This is the case because of the very high FDC coefficients and the concentration of the local optima in a very tiny part of the whole search space. A further confirmation of this impression is given by the analysis of the landscape ruggedness through the correlation length of random walks. In fact, later experimental results suggest that MBLB instances are easily solved, while XLOLIB and LMC-LOP instances of a similar size are by far harder for metaheuristics but also for exact algorithms. An additional, interesting observation is that the landscape analysis would suggest that the characteristics of the XLOLIB instances are different from the original LOLIB instances, differently from the structural analysis of these instances. In fact, the later experimental evaluation showed that XLOLIB instances are much harder to solve than LOLIB instances (see also experimental evaluation in Section 7) and this observation gives an indication that the hardness of XLOLIB instances may not only be due to their larger size.

6. Metaheuristics

The results of the search space analysis of the LOP suggest that methods that are able to exploit both the good performances of the local search, and the often highly positive fitness distance correlation are promising for this problem. Earlier research results suggest that two metaheuristics that have these characteristics are Iterated Local Search (ILS) [19], and Memetic Algorithms (MAs) [5, 21, 20].

6.1. ITERATED LOCAL SEARCH

ILS is a conceptually very simple but at the same time very powerful metaheuristic, as shown by a number of applications [19]. ILS iterates in a particular way over the local search process by applying three main steps: (i) perturb a locally optimal solution, (ii) locally optimize it with the local search chosen and (iii) choose, based on some acceptance criterion, the solution that undergoes the next perturbation phase. Algorithm 1 describes the general algorithmic outline for ILS. Next, we indicate the possibilities we considered for the final ILS algorithm.

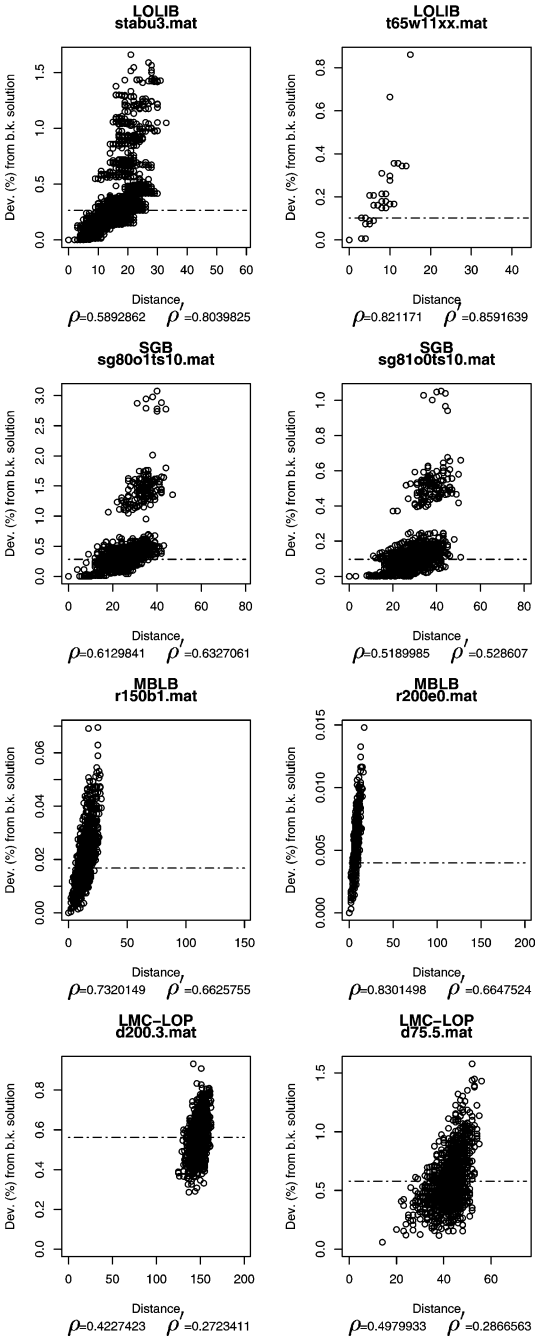

Figure 3. Examples of FDC plots. On the x-axis is given the distance to the nearest global optimum (or best known solution if optima are not proven) and on the y axis the percentage deviation from the optimum or best known solution. The dashed line indicates the median deviation from the best known or globally optimal solution over the randomly generated local optima.

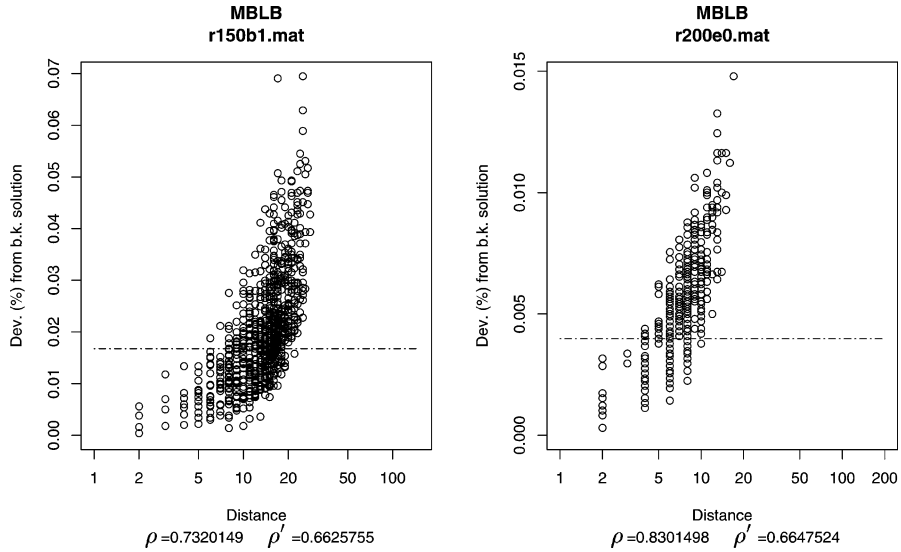

Figure 4. FDC plots for two MBLB instances; here the x-axis uses a log scale.

ALGORITHM 1. Algorithmic outline of an ILS algorithm.

$\pi \leftarrow GenerateInitialSolution()$;
$\pi \leftarrow LocalSearch(\pi)$;
repeat
  $\pi' \leftarrow Perturbation(\pi)$;
  $\pi' \leftarrow LocalSearch(\pi')$;
  $\pi \leftarrow AcceptanceCriterion(\pi, \pi', history)$;
until termination condition met;

- *GenerateInitialSolution*: The initial solution is taken to be a random permutation.
- *LocalSearch*: The local search procedure is the core of the algorithm and the overall ILS performance depends strongly on it. For the ILS algorithm, we use the $\mathcal{LS}_f$ local search, which was the best performing one according to Section 3.
- *Perturbation*: As the perturbation operator we used *interchange* moves, because it is a move that cannot be undone by insert moves in one step. The number of *interchange* moves to be applied in a perturbation is a parameter of the algorithm.
- *AcceptanceCriterion*: The acceptance criterion determines to which solution the next perturbation is applied. We tried different approaches, the final choice of which one to be used was made using an automatic tuning procedure.

  Accept better: A new local optimum is accepted only if the objective function is larger than the current best solution, that is, is $f(\pi') > f(\pi)$;

Accept small worsening: A new local optimum is accepted if the objective function $f(\pi')$ is larger than $(1 - \epsilon) \cdot f(\pi)$, where ϵ is a parameter to be tuned;

Simulated annealing like: A probabilistic acceptance/rejection test is applied, based on the standard Metropolis acceptance criterion in simulated annealing [16]. In this case, the parameters to be tuned are the initial temperature, the temperature cooling ratio, and the number of steps between consecutive temperature reductions.

6.2. MEMETIC ALGORITHM

MAs are evolutionary algorithms that are intimately coupled with local search algorithms, resulting in a population-based algorithm that effectively searches in the space of local optima [24].

ALGORITHM 2. Algorithmic outline of a memetic algorithm.

```
Population ← {};
for i = 1 ... m do {m is the number of individuals}
  π ← LocalSearch(GenerateRandomSolution());
  Population ← Population ∪ {π};
end for
repeat
  Offspring ← {};
  for i ← 1 ... #crossovers do
    draw π^a, π^b from Population
    Offspring ← Offspring ∪ {LocalSearch(Crossover(π^a, π^b))};
  end for
  for i ← 1 ... #mutations do
    draw π^a from Population
    Offspring ← Offspring ∪ {LocalSearch(Mutate(π^a))};
  end for
  Population ← SelectBest(Population ∪ Offspring, m);
  if same average solution quality for a long time then {diversification}
    Population ← SelectBest(Population, 1);
    for i = 1 ... m − 1 do {m is the number of individuals}
      π ← LocalSearch(GenerateRandomSolution());
      Population ← Population ∪ {π};
    end for
  end if
until termination condition met;
```

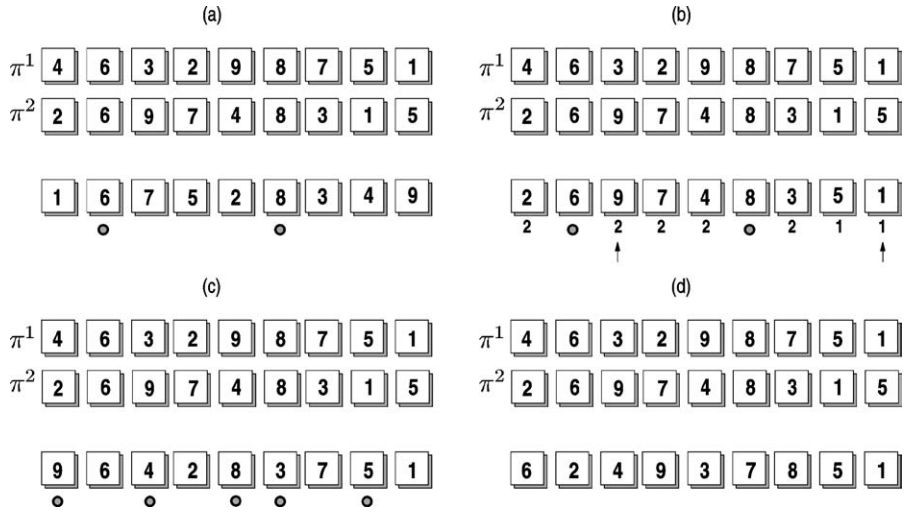

Figure 5. (a) DPX operator: the positions marked with a circle are common to the parents; (b) CX operator: the positions marked with a circle are common to the parents (c) OB operator: the positions marked with a circle are the reordered elements ($k = 5$); (d) Rank operator.

Algorithm 2 shows the algorithmic scheme of MAs that we used in our implementation. In the first step, a *population* of *individuals* is obtained by first generating m distinct random permutations and applying to each $\mathcal{LS}_f$. Then, in each iteration (*generation*) a number of new individuals are created by applying *crossover* and *mutation* operators (in the literature these new individuals are called *offspring*). The individuals to which *crossover* and *mutation* are applied are chosen randomly according to a uniform distribution as in several other, high performing MAs [20]. The crossover operator takes two individuals of the current population and combines them into a new individual, while the mutation operator introduces a perturbation into an individual. To each of the offspring $\mathcal{LS}_f$ is applied. Finally, the best m individuals from the original population and the newly generated ones are selected for the new population; care is taken to eliminate duplicates.

In addition to this rather standard scheme for MAs, we use a diversification mechanism that is triggered if the average objective function value of the population has not changed for a number of steps. In this case, we generate a new, random initial population, keeping only the overall best individual.

It is well known that the performance of an MA may depend strongly on the crossover operator. Therefore, we tested four different ones.

DPX (Figure 5a): The offspring inherits the elements that have the same position in both parents; these are put into the same position as in the parents. The other elements are assigned randomly between those positions that have not yet been chosen. This results in an offspring that, on average, has the same distance from both parents.

CX (Figure 5b): The idea of CX is to keep as much information as possible from the parents. For the elements in common between the parents it works like the previous operator. For the others, the CX operator chooses randomly an empty position (i) and a parent (π^1), determining an element a ($a = \pi_i^1$) that in turn is assigned to the offspring in position i. In the second parent, π^2, a different element $b = \pi_i^2$ occupies this same position i. The element b is then copied to the offspring in the position occupied by b in π^1. This process iterates until all the position are filled (see [22]).

OB (Figure 5c): In the first phase of the order-based crossover, the solution of the first parent is copied to the offspring. In the second phase it selects k positions, $0 < k < n$, and orders the elements in these k positions according to their order in the second parent (see [32]);

Rank (Figure 5d): The offspring permutation is obtained by sorting the elements by their average ranking over the two parents, ties are broken randomly according to a uniform distribution.

6.3. PARAMETER TUNING

The tuning of the ILS and the MA algorithm was done in a systematic, statistically well-founded approach. We have developed a number of different candidate configurations for the two algorithms, 78 in the ILS case and 144 in the MA case, and a final configuration was selected using an automatic tuning procedure based on F-races [4]. The F-race returns the configuration of a metaheuristic, that performs best with respect to some evaluation metric on a number of instances that are used for parameter tuning. In our case, parameter tuning was done using XLOLIB instances of size 100. Notice that the instances used for tuning are different from the solutions in the benchmark sets on which the computational results are presented in the following. Hence, we have a clear separation of the instances into training instances, used for parameter tuning and test instances, on which the final results are presented.

Certainly, it may be argued that tuning the algorithms on one specific instance class and testing them in possibly different ones may give a bias in the results. However, this procedure gives also an impression of the robustness of an algorithm, since we can examine how the performance on one instance class generalizes to a wider set of instances. Additionally, we did further experiments testing different configurations when deemed necessary (see, for example, Section 7.3), so that a more complete picture of the overall performance can be obtained.

In the ILS case, the tuning concerned mainly the acceptance criterion to be used and the strength of the perturbation. The final configuration uses the acceptance criterion that accepts slightly worsening solutions with $\epsilon = 0.0001$ and the

perturbation consists of 7 *interchange* moves. We refer to this ILS algorithm with these parameter settings as ILS-LOP in what follows.

In the MA case, we performed some exploratory experiments before applying the actual tuning procedure. In preliminary experiments we found that the CX and OB crossovers gave best results. Therefore, we considered only these two in the automatic tuning phase. Mutations are obtained by applying the *interchange* operation to an individual. The number of times the operation is applied is a parameter; we fixed this parameter to 7, so that the mutation corresponds to the perturbation in the ILS case. In the actual tuning phase with the F-races we considered 144 different configurations (more than for ILS, but ILS has fewer parameters). We compared the one that resulted to be the best to the two extreme cases that are an MA without crossover and one without mutation. This final comparison led to the choice of the configuration with no mutation, supposedly, because the partial restarts were enough to introduce diversification into the search, making the mutation unnecessary. The final configuration had a population size of 25 individuals, every generation 11 new offsprings are formed using the OB crossover operator, and the diversification was applied after the average fitness of the population has not changed for 30 generations (we refer to this MA as MA-LOP in what follows).

7. Experimental Results

In this section we study the performance of ILS-LOP and MA-LOP and compare them to results from the literature. The algorithms were run on dual processor Athlon 2400+ machine with a clock speed of 2 GHz and with 1 GB of RAM; since the algorithms were implemented as single processes (no threads) only one processor was used at a time. Each algorithm was run 100 times on each instance from LOLIB, MBLB, and SGB and 30 times on the XLOLIB and the LMC-LOP instances. For the experiments a CPU time limit of 120 seconds was fixed, only runs that reached the known optimum solution value were aborted prematurely.*

The performance analysis is divided into two parts, in the first we consider the instances for which we know the optimal or the conjectured optimal objective function value, while in the second the other instances are considered. Next, different aspects of the algorithms are analyzed in more detail and the results are compared to those in the literature.

7.1. INSTANCES WITH KNOWN OPTIMAL SOLUTIONS

For all LOLIB, SGB and MBLB instances we were able to determine the optima, using an exact algorithm that is presented in [23]. The same algorithm is not able to find solutions for the instances of the other classes in reasonable computation

* To give some impression of the time limits: the time taken to apply 1,000 time $\mathcal{LS}_f$ starting from random solutions on the `be75eec_250.mat` instance from XLOLIB of size 250 took 5.86 seconds.

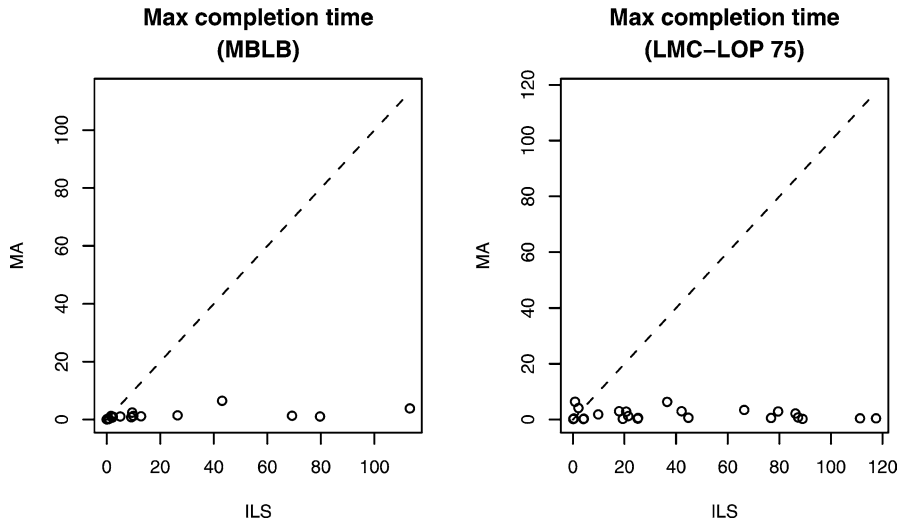

Figure 6. Pairwise comparison of ILS-LOP and MA-LOP with respect to the maximum time taken to solve MBLB (left) and LMC-LOP (right) instances. Each point corresponds to one instance; the x-axis indicates the maximum computation time of ILS-LOP and the y-axis gives the maximum computation time for MA-LOP.

time (this was tested running randomly chosen instances for 18 hours). For LMC-LOP instances of size 75, we observed that MA-LOP found for each instance the same objective function value; the very same value was the best found by ILS-LOP, which, however, did not reach such a solution in every single trial. Therefore, we strongly suppose that this value is optimal and we used it as the conjectured global optimum for our analysis. In fact, the generated solutions were found to be unique. Solutions that we conjecture to be global optima will also be called pseudo-optima in the following.

Experiments with MA-LOP and ILS-LOP showed that the LOLIB and SGB instances are extremely easy and cannot be considered as challenging benchmarks for state-of-the-art metaheuristics. In fact, the longest trial of ILS-LOP and MA-LOP for any of the instances of LOLIB took less than 0.1 seconds and less than 0.2 seconds for the SGB instances.

For MBLB the situation is different. While MA-LOP obtained very good results and the maximum time to solve an instance was 6.5 seconds, the results for ILS-LOP were much worse, as shown in Figure 6 on the left. In fact, for 2 instances ILS-LOP failed to find an optimum in all the runs (once in one instance, three times for the other one) and when it succeeded, it took much longer than MA-LOP.

Somewhat stronger tradeoffs between the performance of the two algorithm became noticeable on the LMC-LOP instances of size 75 (Figure 6, right). Here, MA-LOP obtained much better results than ILS-LOP: First, as said above, MA-LOP was able to find the best known solutions in all the 30 runs, while ILS-

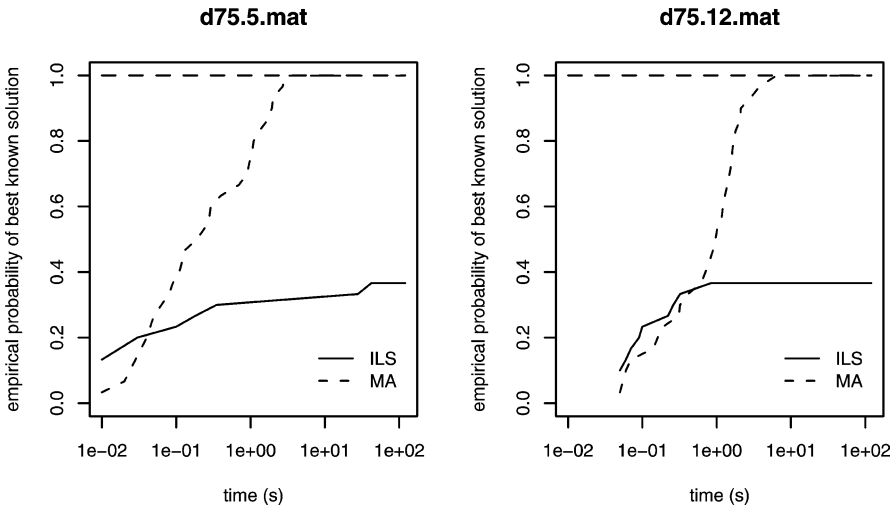

Figure 7. Run time distributions for the hardest LMC-LOP instances of size 75. The computation time is given on the x-axis and the empirical cumulative probability of finding a pseudo-optimal solution is given on the y-axis.

LOP could reach the same level of performance only for 12 instances. For seven instances, it found the best known permutation even in less than 50% of the runs.

To give a more detailed impression of the behavior of MA-LOP and ILS-LOP, for selected instances we examined the run-time distributions that give the empirical probability of finding a global optimum or pseudo-optimum (or a bound on the solution quality) in dependence of the computation time [12]. Figure 7 gives the RTDs for the two hardest LMC-LOP instances of size 75 as judged by the computational results of ILS-LOP. In both cases, ILS-LOP finds the best known solution in only 11 out of 30 trials. The plots show that (i) the probability of finding such a solution for MA-LOP increases continuously and quickly reaches one, suggesting that MA-LOP is preferable to ILS-LOP for computation times in the range of a few seconds and (ii) ILS-LOP suffers from a type of stagnation behavior that strongly compromises its performance.

In fact, the observation of search stagnation suggests that ILS-LOP performance can be strongly improved by including additional means of search diversification [31]. Therefore, we introduced a simple diversification step for ILS-LOP that restarts the algorithm from a new random solution, if no improved solution is found for n_{ni} of iterations. The parameter n_{ni} was set (without tuning) to 750, which is the same number of solutions visited by MA-LOP before the diversification step is applied. The new algorithm (ILS_R) strongly improved over ILS-LOP, although it did not fully reach the level of performance of MA-LOP. Certainly, ILS_R could be improved by additional tuning or by using more sophisticated ways of search diversification than in ILS-LOP.

Table X. For LMC-LOP instance of size 75, where ILS-LOP did not find the best known solution in all trials, we give statistics on the time it reached the last improvement in solution quality. Given that the maximum computation time was 120 seconds, the results indicate that ILS-LOP stagnates very early in the search. No-opt indicates the number of runs (out of 30) for which the best known solution was not found

Instance	Median	Mean	Max	No-opt
d75.1.mat	0.095	2.045	36.52	17
d75.3.mat	1.360	5.730	28.65	16
d75.4.mat	0.335	1.609	11.87	17
d75.5.mat	0.075	3.478	42.13	19
d75.6.mat	0.385	1.131	21.42	12
d75.7.mat	0.010	7.690	111.27	5
d75.9.mat	0.325	6.823	79.57	18
d75.12.mat	0.055	0.208	1.51	19
d75.13.mat	1.595	8.952	86.24	18
d75.14.mat	2.650	26.078	117.41	9
d75.19.mat	0.150	2.385	25.27	2
d75.21.mat	0.920	8.429	87.30	14
d75.23.mat	0.030	0.144	1.25	2

Finally, we compare the computational results of MA-LOP to an exact algorithm by Mitchell and Borchers (SimpMB) [23], which is based on the Simplex algorithm and that uses a branch and bound procedure to find an integer solution. Figure 8 gives a visual comparison of the results on the LOLIB and MBLB instances. MA-LOP is clearly much faster than SimpMB, often by several orders of magnitude. The much superior performance of MA-LOP over SimpMB is further confirmed by experiments run on some XLOLIB and LMC-LOP instances in which SimpMB could not solve these instances within several CPU days.

7.2. INSTANCES WITH UNKNOWN GLOBAL OPTIMA

For XLOLIB and LMC-LOP instances with matrices of dimension larger than 75 no optimal or pseudo-optimal solutions are known. Therefore, we restricted the comparison of MA-LOP and ILS-LOP to a statistical analysis of the quality of the solutions returned by the two algorithms after the maximum computation time. The overall result was that for the large instances of LMC-LOP, MA-LOP obtains average results that are significantly better than those of ILS-LOP, as confirmed by a Wilcoxon test with $\alpha = 0.01$. Similarly, MA-LOP gives better performance on the XLOLIB instances. This is true for the instances of dimension 150, as indicated by

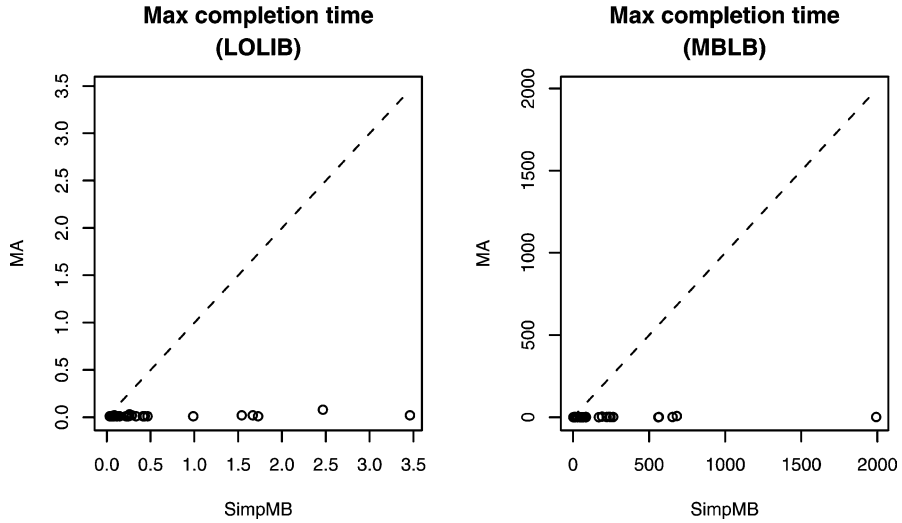

Figure 8. Pairwise comparison of SimpMB (indicated as exact method) to MA-LOP for the LOLIB and MBLB instances. For each instance, the time given for MA-LOP is the maximum time over 100 trials to find a globally optimal solution (y-axis), while the timing for SimbMB (given in the x-axis) is obtained from running it once since it is a deterministic algorithm.

a Wilcoxon test with $\alpha = 0.01$; however, for the large instances of dimensionality 250, no significant differences between MA-LOP and ILS-LOP could be found. The computational results are visualized in Figure 9.

7.3. TUNING EFFECT FOR ILS-LOP

The results presented in Section 7.1 for ILS-LOP on the MBLB instances are actually much worse than those presented in an earlier article with a different ILS algorithm [25] using the $\mathcal{CK}$ local search and different parameter settings for the perturbation size (5 *interchange* moves were used) and the acceptance criterion (only better quality solutions were accepted). Hence, we suspect that the main reason for the "poor" performance of ILS-LOP is that the configuration returned from the automatic tuning procedure performs poorly on MBLB instances. Hence, we tested again this earlier ILS implementation, referring to it in the following as ILSv5, with the only difference that now we use the $\mathcal{LS}_f$ local search instead of $\mathcal{CK}$.

In fact, ILSv5 resulted to be far better than ILS-LOP; it was able to reach the optimal solution in all trials for every instance, and the computation times were even smaller than that of MA-LOP, as shown by Figure 10. The hardest instance was solved by ILSv5 in a maximum computation time of less than 5 seconds, while all other instances took less than one second.

These results suggest, in turn, that ILSv5 may perform better than ILS-LOP also on other instance classes. We tested this conjecture on the LMC-LOP instances

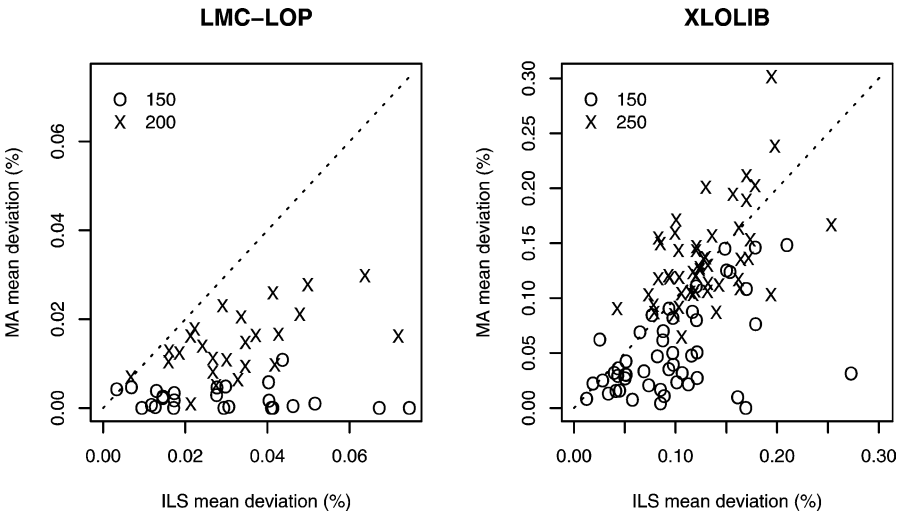

Figure 9. Pairwise comparison of the average solution quality obtained by ILS-LOP and MA-LOP, on large LMC-LOP instances (left) and XLOLIB instances (right). The results are grouped according to the different instance sizes (indicated by crosses or circles). Each cross (circle) gives on the x-axis the average deviation from the best known solution found by ILS-LOP and on the y-axis that of MA-LOP.

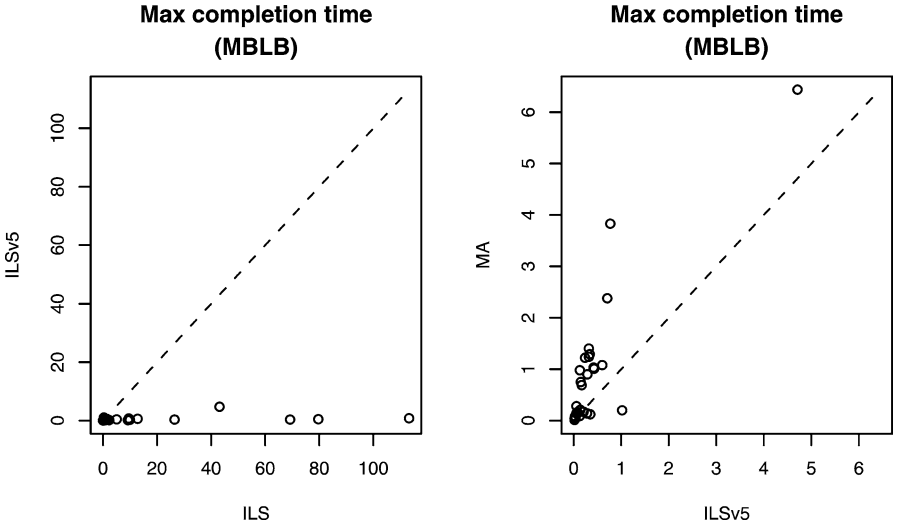

Figure 10. Pairwise comparison of ILSv5 to ILS-LOP (left) and ILSv5 to MA-LOP (right) based on the maximum time measured across 100 trials to find a globally optimal solution on MBLB instances.

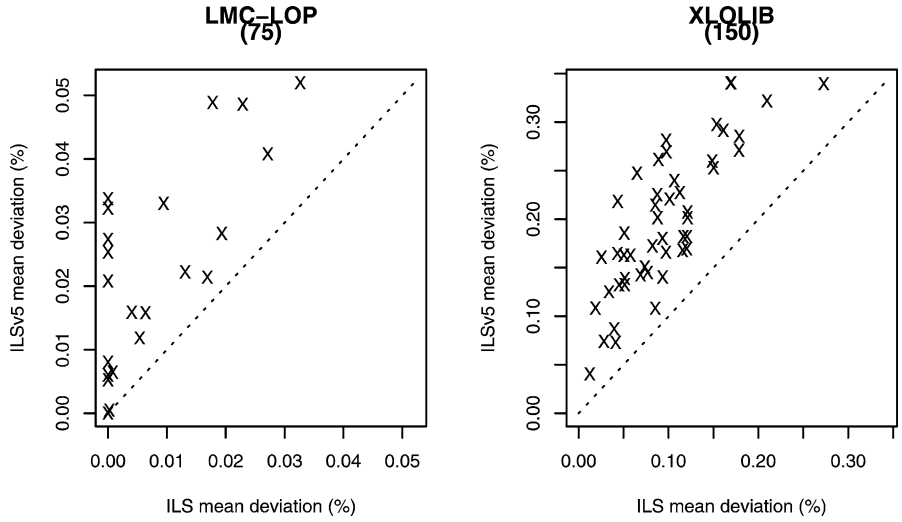

Figure 11. Pairwise comparison between the ILS-LOP and ILSv5 on LMC-LOP and XLOLIB instances. Each cross gives on the x-axis the average deviation from the best known solution found by ILS-LOP and on the y-axis that of ILSv5.

of dimension 75 and the XLOLIB instances of size 150. For these instances, we computed the average deviation from the best known solutions and plotted these in Figure 11; the result is that ILSv5 is significantly worse than ILS-LOP on these instances.

Overall, these results suggest that the performance of MA-LOP is more robust with respect to the various instance classes than ILS-LOP. This conclusion can be drawn because (i) ILS-LOP and MA-LOP were only tuned on the dimension 100 instances from XLOLIB, but MA-LOP shows, in general, good behavior across all instance classes and (ii) variants of ILS-LOP that differ only in some details of the parameter tuning make a large difference to performance and, hence, make parameter settings strongly dependent on instance classes.

7.4. FDC AND INSTANCE HARDNESS

The experiments with ILS-LOP on the LMC-LOP instances indicate that ILS-LOP has significant difficulties for solving all instances in each single run. One reason may be that ILS-LOP is attracted to high quality solutions that may be far from a globally optimal one. But, if this is the case, it is likely that ILS-LOP shows such a behavior on instances with a low FDC value.

This conjecture is examined by analyzing ILS-LOP results in dependence of the FDC coefficient ρ and the easy level FDC coefficient ρ'. As an index of how hard is an instance for ILS-LOP, we used the average deviation from the best known solutions and the number of best known solutions returned by ILS-LOP over 30 trials. The plots in Figure 12 illustrate graphically the relationship of

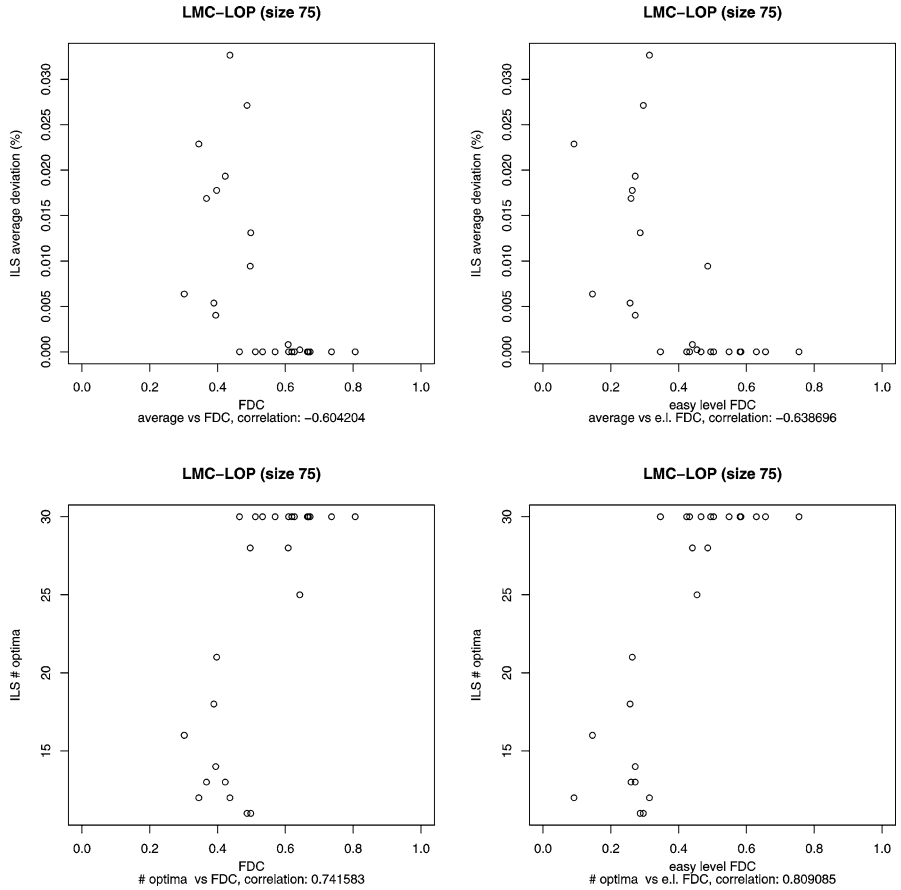

Figure 12. FDC affects the instance hardness. Shown are plots of the average percentage deviation from pseudo-optimal solutions for ILS-LOP (in the first row) and the number of best known solutions found by ILS-LOP (second row) versus the FDC (left column) and the easy level FDC (right column). In addition, are given the correlations between each pair of measures.

these measures to the FDC and easy level FDC and show that these search space characteristics affect the hardness of an instance as encountered by ILS-LOP. The FDC has a strong negative correlation with the average deviation from the optima found by ILS-LOP and a strong positive one with the number of global optima. Summarizing, the higher is the FDC the easier become the instances for ILS-LOP, as we conjectured. A slightly stronger correlation is observed for the *easy level* FDC, which may suggest that the easy level FDC is better suited to predict the instance hardness for ILS-LOP.

Besides the correlation value, the plots illustrate an interesting behaviour: if we consider the FDC ρ, there exists a transition phase between the values 0.4 and 0.7; for smaller values the instances result to be hard, while for larger values all instances are easy. The same happens for the easy level FDC (ρ'), but here the

Table XI. Comparison of ILS-LOP and MA-LOP to three algorithms from the literature. Avg.Dev. (%) gives the average percentage deviation from the known optimal solutions, # optima the number of optimal solutions found, run time (s) gives the run-times reported in the original papers, and "P166 run time" are the computation times translated to a 166 MHz Pentium CPU

		SS	ETS	IDS	ILS-LOP	MA-LOP
LOLIB	Avg.Dev. (%)	0.01	0.00	0.00	0.00	0.00
	# optima	42	47	49	49	49
	Run time (s)	3.82	0.93	1.22	0.00165	0.00176
	P166 run time	3.82	0.93	0.30	0.35	0.37
LMC-LOP size: 75	Avg. Dev. (%)	–	0.05	0.00	0.0072	0.00
	# optima	–	3	25	13 (25)	25
	Run time (s)	–	2.95	10.56	6.58*	0.38
	P166 run time	–	2.95	38.25	138.15	7.90

transition is sharper (between 0.4 and 0.5). This, once more, suggest that ρ' is a better measure to predict the hardness of a given instance.

7.5. COMPARISON

In the literature, we find three main metaheuristic approaches to the LOP. In [7], Campos, Laguna, and Martí proposed the application of Scatter Search (SS) to the LOP and they discussed several ways of how to implement an SS approach to the LOP.* The same authors studied an elite tabu search algorithm with additional diversification features for the LOP [18]. In this latter article, they also presented the instance class LMC-LOP. The most recent metaheuristic application for the LOP is the iterated dynasearch (IDS) of Congram [9]. Dynasearch is a local search algorithm where a dynamic programming approach is used to find the best set of independent *insert* moves (two moves are independent if they do not overlap); iterated dynasearch is then simply an ILS algorithm that uses dynasearch in the local search step.

In the following, we give some comparisons on the solution quality and the timings between the different available approaches. However, we encountered several difficulties for doing so. The least severe probably is that the experiments in these articles were run on different machines. Using the SPEC benchmark results [29] and some experimental test we found that the machine we use is 21 times faster than the Intel Pentium 166 MHz used in [7, 18] and 15 times faster than the Power Challenge R10000 used in [9]. In Table XI we report some results for the instance

* The scatter search of [6] differs only in minor details from the one in [7] and the results are very similar. Therefore, we focus in the following on the results presented in the first article.

Table XII. Comparison of ILS-LOP and MA-LOP to ETS on large LMC-LOP instances. Given is the average percentage deviation from the best known solutions, averaged over all trials and all instances

Size	ETS	ILS-LOP	MA-LOP
150	0.18	0.029	0.0022
200	0.19	0.033	0.015

classes studied in [9, 7, 18]. The major difficulty for the comparison we found were that not enough details were given in these articles to allow a detailed comparison. First, the termination criterion applied to ETS and IDS is not clearly stated, neither how many trials were run on the different available instances. Second, the "average deviation" is the mean of the results over all the instances of the considered class; however it is not clear which results are reported (for example, if it is the mean over the experiments or the best results obtained). Because of these problems it is not possible to establish if the number of optima given in these articles is the number of instances for which the algorithm was able to get at least once a global optimum, always the optimum, or the result after just one run. To be on the most cautious side (that is, to let the reported results appear in the best possible light), we will assume that averages for SS, ETS, and IDS are given as averages of the best solutions found and that the number of optima is the number of instances that are always solved to optimality.

For the results of MA-LOP and ILS-LOP, we report the average deviation computed over all results obtained in 100 runs for LOLIB and 30 runs for LMC-LOP instances. (Note, that for these instances MA-LOP found the optimal or pseudo-optimal solutions in each single trial for all the instances of LOLIB and LMC-LOP.) For the time we indicate the average time to find an optimal solution when it was found. As the number of optima we report how many instances were solved to the optimum in all the considered runs by MA-LOP, while for ILS-LOP we additionally give the number of instances for which a global optimum was found at least once. Regarding the results in Table XI, let us remark the following. In [9] is reported that the IDS was able to obtain always the same best result for the LMC-LOP instances. Therefore, we assumed that this is the very same result we obtained (and, hence, the resulting average deviation of 0.0 for IDS in Table XI from the best known solution). Instead, for ETS we can give precise results, because Rafael Martí sent us a spread-sheet containing the results of ETS for each instance.

The data for the comparison are given in Table XI. From these data it is clear that, even under the cautious assumptions about the results of SS, ETS, and IDS, our ILS and MA are extremely competitive to the earlier proposed metaheuristic

approaches to the LOP. In fact, ILS-LOP and MA-LOP outperform ETS and SS on the LOLIB instances and are roughly on a par with IDS. On the small LMC-LOP instances, ILS-LOP and MA-LOP return much better quality solutions than ETS at, however, higher computation times. ILS-LOP appears to perform slightly worse than IDS on these instances, while MA-LOP solves the small LMC-LOP instances about five times faster than IDS. Next, we compare the average deviation from the best known solutions for the LMC-LOP instances of dimension 150 or 200 for ETS, ILS-LOP, and MA-LOP in Table XII. (Note that ETS results were adjusted to the new best known solutions for these instances.) The results show that ILS-LOP and MA-LOP yield by far better quality solutions than ETS; however, it is not clear how the computation times of the three algorithms compare, because not enough details are given in [18]. Recently, another memetic algorithm for the LOP has been presented [13]. This algorithm is run once for each LOP, MBLB and LMC-LOP instance; the computational results with that algorithm on LOP and MBLB are comparable to ours; on the LMC-LOP instances they use a very different stopping criterion resulting in much shorter computation times than ours. The solution quality we reach after two minutes is much better than those of [13] for these instances. If we impose similar short computation times for our algorithm, considering the differences in CPU speed, the two MAs seem to be similarly effective.

The overall result of the comparison is that, in particular, MA-LOP obtains excellent performance, from a computation time and solution quality perspective. In fact, even when being cautious about the experimental conditions used in the other papers, our results suggest that MA-LOB is a new, very robust state-of-the-art algorithm for the LOP.

8. Conclusions

In this paper we have given a detailed analysis of benchmark instances for the LOP. These include a new class of instances, called XLOLIB.* The instances of this class are randomly generated through sampling real-world instances, which allows us to derive large, random real-world like instances. In fact, cross-statistical data on the distribution of the matrix entries of XLOLIB instances are basically the same as those of the underlying real-world instances from LOLIB. However, the search space analysis showed some discrepancy between the original LOLIB instances and the newly generated XLOLIB instances. Nevertheless, XLOLIB instances appear to be much closer to real-world instances than instances from other, randomly generated classes like MBLB or LMC-LOP instances.

The search space analysis of LOP instances showed that most instances have high correlation length, suggesting that, in general, the LOP is easy to solve when compared to other problems [2]. Furthermore, most LOP instances have a high fitness distance correlation. Notable exceptions occur for a few LOLIB instances,

* All the instances and the best known results will be published on the WWW at the address `http://intellektik.informatik.tu-darmstadt.de/~schiavin/lop` for their easy access.

where even negative fitness distance correlations were found. Concerning measures of search space characteristics, we introduced a new way to measure the FDC, which we called "easy level FDC". This measure tries to consider the fact that metaheuristics actually search through high quality local optima and the central idea of the easy level FDC is to focus the analysis on high quality solutions. In fact, the easy level FDC showed a better correlation to the instance hardness of small LMC-LOP instances for ILS algorithms than the standard way of determining FDCs.

Based on the results of the search space analysis and the high solution quality returned by simple iterative improvement algorithms, we further studied efficient iterated local search and memetic algorithms for the LOP. A comparison between the two algorithmic approaches showed that the MA resulted in a much more robust performance with respect to the different instance classes than the ILS algorithm. It is an open question whether, with appropriate tuning, ILS can reach the MA's performance on all the instance classes. A final comparison of MA and ILS performance to other available metaheuristic approaches to the LOP showed that our MA is a new, very robust state-of-the-art algorithm for the LOP.

Acknowledgements

The authors would wish to thank John Mitchell and Brian Borchers for making available the code of their exact algorithm. We also thank Rafael Martí for sending to us the LMC-LOP instances and the results of the Tabu Search. Finally, we want thank Joshua Knowles for helpful advices.

This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. Angel, E. and Zissimopoulos, V.: Autocorrelation coefficient for the graph bipartitioning problem, *Theoret. Comput. Sci.* **191**(1–2) (1998), 229–243.
2. Angel, E. and Zissimopoulos, V.: On the classification of NP-complete problems in terms of their correlation coefficient, *Discrete Appl. Math.* **99**(1–3) (2000), 261–277.
3. Becker, O.: Das Helmstädtersche Reihenfolgeproblem – die Effizienz verschiedener Näherungsverfahren, in *Computer Uses in the Social Science*, Wien, January 1967.
4. Birattari, M., Stützle, T., Paquete, L. and Varrentrapp, K.: A racing algorithm for configuring metaheuristics, in W. B. Langdon et al. (eds), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, Morgan Kaufmann Publishers, San Francisco, CA, 2002, pp. 11–18.
5. Boese, K. D.: Models for iterative global optimization, PhD thesis, University of California, Computer Science Department, Los Angeles, CA, 1996.

6. Campos, V., Glover, F., Laguna, M. and Martí, R.: An experimental evaluation of a scatter search for the linear ordering problem, *J. Global Optim.* **21**(4) (2001), 397–414.
7. Campos, V., Laguna, M. and Martí, R.: Scatter search for the linear ordering problem, in D. Corne, M. Dorigo and F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, London, UK, 1999, pp. 331–339.
8. Chanas, S. and Kobylanski, P.: A new heuristic algorithm solving the linear ordering problem, *Comput. Optim. Appl.* **6** (1996), 191–205.
9. Congram, R. K.: Polynomially searchable exponential neighbourhoods for sequencing problems in combinatorial optimisation, PhD thesis, University of Southampton, Faculty of Mathematical Studies, UK, 2000.
10. Grötschel, M., Jünger, M. and Reinelt, G.: A cutting plane algorithm for the linear ordering problem, *Oper. Res.* **32**(6) (1984), 1195–1220.
11. Grötschel, M., Jünger, M. and Reinelt, G.: Optimal triangulation of large real world input–output matrices, *Statistische Hefte* **25** (1984), 261–295.
12. Hoos, H. H. and Stützle, T.: Evaluating Las Vegas algorithms — pitfalls and remedies, in G. F. Cooper and S. Moral (eds), *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, CA, 1998, pp. 238–245.
13. Huang, G. and Lim, A.: Designing a hybrid genetic algorithm for the linear ordering problem, in E. Cantú-Paz et al. (eds), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, Lecture Notes in Comput. Sci. 2723, Springer-Verlag, Berlin, 2003, pp. 1053–1064.
14. Jones, T. and Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in L. J. Eshelman (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA, 1995, pp. 184–192.
15. Kaas, R.: A branch and bound algorithm for the acyclic subgraph problem, *European J. Oper. Res.* **8** (1981), 355–362.
16. Kirkpatrick, S., Gelatt, C. D., Jr. and Vecchi, M. P.: Optimization by simulated annealing, *Science* **220** (1983), 671–680.
17. Knuth, D. E.: *The Stanford GraphBase: A Platform for Combinatorial Computing*, Addison-Wesley, New York, 1993.
18. Laguna, M., Martí, R. and Campos, V.: Intensification and diversification with elite tabu search solutions for the linear ordering problem, *Comput. Oper. Res.* **26**(12) (1999), 1217–1230.
19. Lourenço, H. R., Martin, O. and Stützle, T.: Iterated local search, in F. Glover and G. Kochenberger (eds), *Handbook of Metaheuristics*, International Series in Operations Research & Management Science 57, Kluwer Academic Publishers, Norwell, MA, 2002, pp. 321–353.
20. Merz, P.: Memetic algorithms for combinatorial optimization problems: Fitness landscapes and effective search strategies, PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 2000.
21. Merz, P. and Freisleben, B.: Fitness landscapes and memetic algorithm design, in D. Corne, M. Dorigo and F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, London, UK, 1999, pp. 245–260.
22. Merz, P. and Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem, *IEEE Trans. Evolut. Comput.* **4**(4) (2000), 337–352.
23. Mitchell, J. E. and Borchers, B.: Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm, in H. L. Frenk, K. Roos, T. Terlaky and S. Zhang (eds), *High Performance Optimization*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000, pp. 349–366.
24. Moscato, P. and Cotta, C.: A gentle introduction to memetic algorithms, in F. Glover and G. Kochenberger (eds), *Handbook of Metaheuristics*, International Series in Operations Research & Management Science 57, Kluwer Academic Publishers, Norwell, MA, 2002, pp. 105–144.

25. Schiavinotto, T. and Stützle, T.: Search space analysis of the linear ordering problem, in G. R. Raidl et al. (eds), *Applications of Evolutionary Computing*, Lecture Notes in Comput. Sci. 2611, Springer-Verlag, Berlin, 2003, pp. 322–333.
26. Stadler, P. F.: Towards a theory of landscapes, in R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck and F. Zertuche (eds), *Complex Systems and Binary Networks*, Lecture Notes in Phys. 461, Springer-Verlag, Berlin, 1995, pp. 77–163.
27. Stadler, P. F.: Landscapes and their correlation functions, *J. Math. Chemistry* **20**(1) (1996), 1–45.
28. Stadler, P. F. and Schnabl, W.: The landscape of the travelling salesman problem, *Phys. Lett. A* **161** (1992), 337–344.
29. Standard Performance Evaluation Corporation: SPEC CPU95 and CPU2000 Benchmarks, `http://www.spec.org/`, November 2002.
30. Stützle, T. and Hoos, H. H.: $\mathcal{MAX}$–$\mathcal{MIN}$ ant system, *Future Generation Computer Systems* **16**(8) (2000), 889–914.
31. Stützle, T. and Hoos, H. H.: Analysing the run-time behaviour of iterated local search for the travelling salesman problem, in P. Hansen and C. C. Ribeiro (eds), *Essays and Surveys on Metaheuristics*, Kluwer Academic Publishers, Boston, MA, 2001, pp. 589–611.
32. Syswerda, G.: Schedule optimization using genetic algorithms, in L. Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1990.
33. Weinberger, E. D.: Correlated and uncorrelated fitness landscapes and how to tell the difference, *Biological Cybernetics* **63**(5) (1990), 325–336.